

9/PRTS.

10/507114

PCT/GB03/01096

Key management protocol 0109 Rec'd PCT/PTO 10 SEP 2004

The present invention relates to the distribution and management of session keys in a communications network, for example, an internet broadcast application.

Recent interest in group communications with a very large set of receivers has led to a need for secure communications systems that scale efficiently as the number of users increases. For example, developers of Internet broadcast applications such as teleconferencing and video-on-demand desire more effective secure communication between very large numbers of users.

In group communications, special problems arise in a dynamic group in which new members can join the group and current members can leave the group, either voluntarily or by being ejected. There are at least three security issues that to be considered:

20

1. Group key security (a group key being a key which allows access to information by all the members of the group). It should be computationally infeasible for a person outside the group to discover the group key.

25

2. Forward Security. A system has forward security if a member leaving the group cannot get access to later group keys and so cannot decrypt data sent after that user has left the group.

30

3. Backward Security. A system has backward security if a member joining the group cannot get access to earlier group keys and so cannot decrypt data sent before that user joined the group.

A simple multi-user system provides a key distribution centre (or key server) that is in direct contact with every member of the group. Each member shares a key with the key ,  
5 distribution centre (the member's individual key) and, for group communications, all members share a group key. Each time a member joins or leaves the group, the group key must be updated to ensure backward or forward security as the case may be. When a new member joins the group, the new group key  
10 is sent to the new member, encrypted using the new member's individual key and is sent, as a broadcast, to all existing members, encrypted using the previous group key. Thus a join event is relatively straightforward and scales well in terms of computational effort, broadcast bandwidth requirements and  
15 secure unicast requirements as the number of users increases.

When a member leaves the group, the new group key must be individually sent to members using that member's individual key since, if the new group key was encrypted using the  
20 previous group key, the user that has just left the group would be able to generate that new group key (it being assumed that that user would receive the encrypted new key by permitted means or otherwise).

25 It can be seen that, the computational and communication requirements scale in a linear manner with the number of users. Thus, in a system with a very large number of users, the computational and communication requirements when a member leaves the group can become prohibitive.

30

It can be seen that there is a need to provide a key management system that scales effectively as the number of users increases. In particular, there is a need for a key management system in which the computational time of the

server and the users, the memory storage requirements of the users and the broadcast bandwidth requirements all scale effectively as the number of users increases.

- 5 A hierarchical key tree is disclosed in "Key Management for Multicast: Issues and Architectures" by D Wallner et. al. (National Security Agency, June 1999, [www.ietf.org/rfc/rfc2627.txt](http://www.ietf.org/rfc/rfc2627.txt)).
- 10 A hierarchical binary tree is an efficient tree-based key management technique. A hierarchical binary tree works as follows. A multicast group has  $N$  members ( $M_1$  to  $M_N$ ). A new member joins the group by contacting the controller via a secure unicast channel. At the time the new member joins,
- 15 the new member and the controller negotiate a pairwise secret key.

The controller stores a binary tree structure in which each node contains a key. At the leaves of the tree there are the

20  $N$  secret keys that the controller has negotiated with each of the members of the group. Each member stores a subset of the controller's keys. The subset of keys stored by a member is the set of keys in the path from the leaf to the root of the tree including the leaf and the root itself. The root node

25 represents the key used to encrypt data during the group communication; all other keys in the tree are auxiliary keys used only to facilitate efficient key updates.

Figure 1 shows a hierarchical tree for a system having three

30 users,  $M_1$ ,  $M_2$  and  $M_3$ . The tree has a root node  $K_{14}$  connected to two nodes  $K_{12}$  and  $K_{34}$ .  $K_{12}$  in turn is connected to nodes  $K_1$  and  $K_2$ . Node  $K_{34}$  is connected to node  $K_3$ . The users  $M_1$ ,  $M_2$  and  $M_3$  are associated with nodes  $K_1$ ,  $K_2$  and  $K_3$

respectively. Each of the nodes K1, K2, K3, K12, K34 and K14 represents a cryptographic key.

In a hierarchical tree structure, each member of the group  
5 knows all the keys from its leaf node up to the root node.  
Thus, user M1 knows the keys for nodes K1, K12 and K14. User  
M2 knows the keys for nodes K2, K12 and K14. User M3 knows  
the keys for nodes K3, K34 and K14.

10 Thus, every user knows the key at the root node K14.  
Accordingly, the root key can be used to encrypt all  
transmissions involving users M1, M2 and M3.

If a new user M4 joins the group, that user must be added to  
15 the hierarchical tree. Figure 2 shows the same hierarchical  
tree as Figure 1, except that nodes K14 and K34 have been  
replaced with nodes K14' and K34' and the new user M4 is  
attached via new node K4 to node K34'. The keys K14' and  
K34' are different from the previous keys K14 and K34 to  
20 ensure that the system has backward security. This is  
implemented by the key server at the root node. Key 4 is  
generated by the key server and keys K34 and K14 are updated  
(to K34' and K14' respectively) by the key server.

25 The new user M4 needs to know the keys K4, K34' and K14'.  
This information is transmitted to M4 via a secure channel.

The key server informs the other members of the group of the  
new keys by sending encrypted broadcasts that all members can  
30 receive (non-members will be able to receive the broadcast  
but they will not be able to decrypt the information sent).  
The following broadcasts are made: K34' encrypted with K3,  
K14' encrypted with K34' and K14' encrypted with K12.

User M3 knows the key K3 and can therefore decrypt K34' encrypted with K3 to arrive at K34'. From this, user M3 can decrypt K14' encrypted with K34'. Similarly, users M1 and M2 both know key K12 and can therefore decrypt K14' encrypted with K12. Thus all users once again know all of the keys from their leaf of the tree to the root. Transmissions involving the members of the group (now including the new member M4) can be encrypted with the new root key K14'.

- 10 If user M3 leaves the group, that user must be removed from the hierarchical tree. Figure 3 shows the hierarchical tree of Figure 2, except that user M3 and node K3 have been removed from the tree, and nodes K14' and K34' have been updated to K14'' and K34'' respectively. Thus all of the  
15 keys that were known to M3 (K3, K34' and K14') have been either removed or updated. Thus the system has forward security.

The key server updates keys K14' and K34' to generate keys  
20 K14'' and K34'' respectively. The key server then broadcasts K34'' encrypted with K4 and K14'' encrypted with K34''. The user M4 knows key K4 and so can decrypt K34'' encrypted with K4 to arrive at K34''. Similarly, M4 can decrypt K14'' encrypted with K34'' to arrive at the new root key K14''. As  
25 before, K14'' must also be broadcast encrypted with K12 so that users M1 and M2 can obtain the new root key. Since previous user M3 did not know either key K4 or key K12, he cannot obtain key K14'' from the broadcast messages.

- 30 The principal advantage associated with the use of a tree for the organisation of users in a multi-user system is that any individual user only knows a subset of the keys of the system. Thus, when a user leaves the group, only that subset needs to be updated to ensure backward security. When a user

leaves the group, the number of keys that have to be updated is of the order of  $\log(N)$ , where  $N$  is the number of users. Thus, the number of transmissions required to re-key the tree scales as the number of users increases.

5

It is not essential that a hierarchical tree is a binary tree. A  $P$ -ary tree can be used. As the value  $P$  rises, the storage requirement for each user decreases, but at the expense of an increase in the number of transmissions

10 required from the key server.

A variant of the hierarchical tree described above is the one-way function tree described in "Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized  
15 Initialization" by D Baleson et. al. (TIS Labs at Network Associates, 26 February 1999).

The one-way function tree described by Baleson et. al. is a binary tree. Each node of the tree is associated with two  
20 keys: an unblinded key  $K(x)$  and a blinded key  $K'(x)$ . The session key that is used to encrypt application data (such as a video broadcast) includes both the blinded and unblinded keys of the root node. The blinded key  $K'(x)$  is derived from the unblinded key  $K(x)$  using a one-way function (see below).  
25  $K'(x)$  is 'blinded' in the sense that it is computationally infeasible to find  $K(x)$  from  $K'(x)$ .

Each node in the hierarchical tree (except the leaf nodes) has two children:  $x_{\text{left}}$  and  $x_{\text{right}}$ . The parent node  $K(x)$   
30 is defined by the following formula:

$$K(x) = K'(x_{\text{left}}) \text{ XOR } K'(x_{\text{right}})$$

The members of the system are associated with the leaves of the tree. Each member knows the blinded keys for every node that is a sibling of any of the nodes on the branch of the tree extending from the user to the root of the tree.

5

Taking the binary tree of Figure 1 as an example, the user M1 would know the blinded and unblinded keys for node K1 (K1 and K'1) and would know the blinded keys for nodes K2 (the sibling of K1) and K34 (the sibling of K12) (the keys K'2 and K'34 respectively). From this information, the user M1 can generate the unblinded key for K12 from the blinded keys K'1 and K'2 thus:

$$K_{12} = K'1 \text{ XOR } K'2$$

15

Using a one-way function generates the blinded key K'12 of node K12 (K'12) with the result that the twin keys (blinded and unblinded) of K12 (K12 and K'12 respectively) are generated. Further, user M1 can generate the unblinded key of node K14 from the blinded keys K'12 and K'34 thus:

$$K_{14} = K'12 \text{ XOR } K'34$$

Using a one-way function generates the blinded key of K14 (K'14) so that the twin keys of the root node K14 (K14 and K14') are known.

The purpose of the blinded and unblinded keys is the reduction of the number of keys that a key distribution server has to send during key update operations. The key distribution server must send  $\log_2(N)$  updates in the form of blinded keys (where N is the number of users). The updates are encrypted to ensure that only the members who should

receive the updates have the necessary keys to decrypt the messages and receive the updates.

One-way functions such as that used in the one-way function tree described above are mathematical functions that are relatively easy to compute in a first direction but is computationally infeasible to compute in the other (reverse) direction.

10 Message digest, fingerprint or compression functions are examples of a first class of one-way functions (functions of this class are commonly called "hash functions"). A message digest function is a mathematical function that takes a variable length input string and converts it into a fixed-length binary sequence. Modern message digest functions typically produce hash values of 128 bits or longer.

Message digest functions are used to create a digital signature for a document. Since it is computationally infeasible to deliberately produce a document that will hash to a particular hash value and extremely unlikely to find two documents that hash to the same value, a document's hash value can serve as a cryptographic equivalent of the document.

25

Examples of message digest functions are MD4 (Message Digest 4), MD5 (Message Digest 5, see "The MD5 Message-Digest Algorithm" by R. Rivest, MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992, [www.ietf.org/rfc/rfc1321.txt](http://www.ietf.org/rfc/rfc1321.txt)) and SHA (Secure Hash Algorithm). SHA is generally considered to be the most secure of the three.



One-way functions can also be generated using pseudo random function (PRF) with varying input and output lengths. A suitable known PRF is an encryption algorithm called RC5. The RC5 encryption algorithm is a fast symmetric cipher  
5 algorithm suitable for hardware or software implementation and has low memory and computational requirements.

Another example of a one-way function is a trapdoor one-way function. The inverse of a trapdoor one-way function is  
10 easily generated if the trapdoor is known but difficult otherwise.

A public-key cryptosystem can be designed using a trapdoor one-way function. Public-key cryptosystems are well known in  
15 the art (see Digital Communications Fundamentals and Applications, Bernard Sklar, Prentice-Hall International, Inc., 1998 edition, pages 698 to 702). The public key in such a system gives information about the particular instance of the function; a private key gives information about the  
20 trapdoor. The function can be computed in the forward direction only unless the trapdoor is known. The forward direction is used for encryption and signature verification. The reverse direction is used for decryption and signature generation.

25 The prior art has addressed some of the problems associated with the distribution and management of session keys in a communications network. In particular, the use of hierarchical trees provides systems in which bandwidth usage  
30 and key storage by the key distribution server scales logarithmically as the number of users increases.

There are problems with the prior art systems. For example, the algorithms described all require the update information

to be encrypted in such a manner that only members entitled to the update information have the necessary keys to decrypt that information.

5 According to the present invention there is provided a method of managing keys in a key distribution system for a communications group, the key distribution system maintaining a tree of nodes including at least one leaf node that has a parent node, each node of the group being associated with a  
10 first key,

the method comprising:

the system updating the first keys of a first branch of nodes in the tree by allocating new first keys to each of the nodes in the branch;

15 the system determining an offset for generating the updated first key of each node in the branch from the previous node in the branch; and  
broadcasting each of said offsets so that, given the updated first key associated with the first node of said branch, each  
20 updated first key of said branch of nodes can be calculated.

Preferably, the first key of each parent node in said tree of nodes is generated from the first key of each of its child nodes by two one-way functions and a mixing function, the  
25 mixing function including the offset as a parameter.

The present invention further provides a key distribution system for a communications group, the key distribution system maintaining a tree of nodes including at least one  
30 leaf node that has a parent node, each node being associated with a first key, wherein:  
the first key of each parent node in the tree is derived from the first key of each of its child node by two one-way

functions and a mixing function, the mixing function including an offset value as a parameter.

The present invention also provides a key distribution system for a communications group, the key distribution system comprising an encryption key and maintaining a tree of nodes including a root node that has at least one child node, and at least one leaf node that has a parent node, the communication group comprising at least one member, wherein the encryption key comprises a join field and a leave field, and wherein:

each member of the group knows the join field of the encryption key;

each node of the key distribution system is associated with a leave key;

the leave field of the encryption key is derived from the leave key of the root node.

Further preferred features of the invention are set out in the appended claims.

From the description of the below it will become apparent that the one-way functions provide the security for the key distribution, while the offset message mechanism make the distribution more efficient.

A protocol for the distribution and management of session keys in a communications network will now be described by way of example with reference to the accompanying drawings, in which

FIGURE 1 shows a hierarchical tree having three users;

FIGURE 2 shows the hierarchical tree of Figure 1 with the addition of a fourth user;

FIGURE 3 shows the hierarchical tree of Figure 2 after one of the four users has been removed;

FIGURE 4 demonstrates the generation of related chains of one-way functions;

5 FIGURE 5 demonstrates the generation of related chains of double one-way functions;

FIGURE 6 shows a hierarchical tree used in the present invention;

FIGURE 7 demonstrates the generation of keys in related  
10 chains of one-way functions in accordance with the present invention;

FIGURE 8 demonstrates the generation of keys from offset messages, in accordance with the present invention;

FIGURE 9 demonstrates the addition of a new member into the  
15 hierarchical tree structure of Figure 6;

FIGURE 10 is a flow chart showing the process by which a new member joins the group;

FIGURE 11 demonstrates the removal of a member from the hierarchical tree structure of Figure 9;

20 FIGURE 12 is a flow chart showing the process by which a member leaves the group.

The preferred embodiment of the present invention uses what will be called an offset hierarchy binary tree (OHBT). The  
25 OHBT system uses offset messages to implement the key update and key recovery mechanisms. The offset message can be considered to be the distance between two chains of one-way functions. Figure 4 shows a first chain having keys X0, X1, X2, X3 and X4. X1 is generated from X0 using a one-way  
30 function f, X2 is generated from X1 using one-way function f, X3 is generated from X2 using one-way function f and X4 is generated from X3 using one-way function f. Figure 4 also shows a second chain having keys Y0, Y1, Y2, Y3 and Y4. In a

similar manner to the first chain, the keys Y0 to Y4 are each separated by one-way function f.

X0 and Y0 are unrelated different keys. However, it is possible to move from one chain to the other using a straightforward formula. The inventor has noticed that, given Y1, X2 can be generated using the following formula:

$$X2 = f(Y1 \text{ XOR } \text{Offset})$$

10

where

$$\text{Offset} = X1 \text{ XOR } Y1$$

15 Since

$$\begin{aligned} f(Y1 \text{ XOR } \text{Offset}) &= f[Y1 \text{ XOR } (X1 \text{ XOR } Y1)] \\ &= f[Y1 \text{ XOR } (Y1 \text{ XOR } X1)] \\ &= f[(Y1 \text{ XOR } Y1) \text{ XOR } X1] \\ &= f(0 \text{ XOR } X1) \\ &= f(X1) \\ &= X2 \end{aligned}$$

20

In this manner, the user knowing the root key Y0 of chain Y can, given the correct offset message, recover X2 and, from X2, he can recover the later keys in the X chain.

25

There is a lack of security in the offset system described above in that, from Y1 and the offset to generate X2 the user can also generate X1, since that offset is simply X1 XOR Y1.

30

If the chains X and Y are keys in a cryptosystem, then this lack of security is not acceptable since, in order to calculate key X2, the user knowing chain Y can generate a

key, X1, that he should not be given access to, thereby disclosing a key that should be confidential.

A solution to the lack of security described above is to  
5 generate intermediate keys. Figure 5 shows a first chain having keys X0, X1, X2 and X3 and intermediate keys f(X0), f(X1) and f(X2). f(X0) is generated from X0 using a one-way function, X1 is generated from f(X0) using one-way function f, f(X1) is generated from X1 using one-way function f, X2 is  
10 generated from f(X1) using one-way function v, f(X2) is generated from X2 using one-way function f and X3 is generated from f(X2) using one-way function f. Figure 5 also shows a second chain having keys Y0, Y1, Y2 and Y3 and intermediate keys f(Y0), f(Y1) and f(Y2). In a similar  
15 manner to the first chain, Y0, f(Y0), Y1, f(Y1), Y2 and f(Y2) are each separated by a one-way function f.

A user knowing Y0 can generate the key X2 using the following formula:

20

$$X2 = f[f(Y1) \text{ XOR Offset}]$$

where

Offset = f(X1) XOR f(Y1), and

$$25 \quad Y(1) = f[f(Y0)]$$

Accordingly, the user can generate X2 and the later keys in that chain but cannot generate X1. Only the intermediate key H(X1) can be generated and that intermediate key is only used  
30 temporarily in generating the key X2. It is, of course, computationally infeasible to generate X1 from H(X1).

A preferred embodiment of the present invention is for use by a multicast group having N members, M<sub>1</sub> to M<sub>N</sub> and having a

group controller (preferably centralised) called the key distribution server. The group of users are organised in a tree structure as shown in Figure 6. In Figure 6, the users are organised in a hierarchical binary tree having a root node K14 (the key distribution server). Root node K14 has two children, nodes K12 and K34. Node K12 in turn has two children, K1 and K2: node K34 has two children, K3 and K4. The users are associated with the leaves of the tree. In the example of Figure 6, four users  $M_1$  to  $M_4$  are associated with nodes K1 to K4 respectively.

The system transfers application data in a secure manner by encrypting data using a session key. The session key comprises two components: a join field and a leave field.

15 The join field value is common to all members. Each node of the tree is however allocated a different value for its leave field.

The leave field for each node is calculated by the users in a bottom-up approach i.e. given the leave field of a node (the "child\_key"), a user can generate the leave field of the parent of that node in the tree (the "parent\_key"). The calculation of the leave field of a parent node is achieved using the formula:

25

$$\text{Parent\_key} = f(f(\text{child\_key}, \text{seq\_n}, \text{pos\#\_key}) \text{ XOR Offset}).$$

wherein:

30 (a, b, c) represents a appended by b appended by c.

f represents some particular one-way function.

seq\_n is a sequence number. It is increased each time the user has to generate the leave key (see below).

pos#key is the position in the tree of the key that is generated, e.g, node K34 has position 34.

Offset is an offset message, as described above.

In a preferred embodiment of the invention, the formula

Parent\_key = f(f(child\_key, seq\_n, pos#\_key) XOR Offset)

is implemented as:

Parent\_key = f(A XOR offset)

where

A = f[child\_key XOR opad, H(child\_key, seq\_n, pos#\_key)]

and

ipad = the byte 0x36 repeated B times

opad = the byte 0x5C repeated B times

(0x represents a hexadecimal number)

Given the leave field of K1, user M1 can generate the leave field of K12 and, from the leave field of K12, the user M1 can generate the leave field of K14 as follows:

K12 = f(f(K1, seq\_n, 12) XOR Offset\_1)

K14 = f(f(K12, seq\_n, 14) XOR Offset\_12)



Similarly, given K2, K3 and K4, the following leave fields can be calculated by the users M2, M3 and M4 respectively:

```

K12 = f(f(K2, seq_n, 12) XOR Offset_2)
5 K14 = f(f(K12, seq_n, 14) XOR Offset_12)

K34 = f(f(K3, seq_n, 34) XOR Offset_3)
K14 = f(f(K34, seq_n, 14) XOR Offset_34)

10 K34 = f(f(K4, seq_n, 34) XOR Offset_4)
K14 = f(f(K34, seq_n, 14) XOR Offset_34)

```

The generation of the keys K12, K34 and K14 from K1, K2, K3 and K4, including the generation of the intermediate keys is shown in Figure 7. It can be seen that the leave field values are generated using a double one-way function arrangement similar to that described with reference to Figure 5.

20 The leave field K1 is operated on by a one-way function f to obtain f(K1) and then mixed, using an XOR function, with an offset value, offset\_1, to obtain the leave field K12, that can be considered to be part of a different chain of one-way functions. Thus, nodes K1 and K12 of the tree can be

25 considered to be part of two different chains of one-way functions in the same way as nodes Y1 and X2 in Figure 5 are part of different chains of one-way functions. Another similarity between the chain of one-way functions in Figures 5 and 7 is of course that moving from Y1 to X2 in Figure 5 is

30 done via an intermediate key f(Y1) and moving from node K1 to node K12 in Figure 7 is done via an intermediate key f(K1).

In Figure 7, the leave field K2 is operated on by a one-way function f to obtain f(K2) and then mixed, using an XOR

function, with an offset value,  $\text{offset}_2$ , to obtain the leave field  $K_{12}$ , that can be considered to be part of a different chain of one-way functions. The leave field  $K_{12}$  calculated is, of course, the same as the leave field  $K_{12}$  referred to  
5 above.

In a similar manner, Figure 7 shows the generation of leave field  $K_{34}$  from leave field  $K_3$  via an intermediate key  $f(K_3)$  and a mixing function and from leave field  $K_4$  via an  
10 intermediate key  $f(K_4)$  and a mixing function.

The leave fields  $K_{12}$  and  $K_{34}$  are operated on by a one-way function  $f$  to obtain intermediate keys  $f(K_{12})$  and  $f(K_{34})$  respectively. Those intermediate keys are mixed, using an  
15 XOR function, with offset values  $\text{offset}_{12}$  and  $\text{offset}_{34}$  respectively, to obtain the leave field  $K_{14}$ .

In order to initialise the hierarchical tree of figure 6, the key distribution server shares a different secret key with  
20 each user of the system. This may be achieved by the key server distributing a certificate containing its public key. A user that wants to be part of the group then sends to the key server a random key (the secret key that will be shared between the user and the key distribution server) that is  
25 encrypted using the public key. The key server decrypts the encrypted message to regenerate the random key.

Refer to Figure 6. Assume that the user  $M_1$  shares a secret key  $K_{m1}$  with the key distribution server. The user  $M_1$  is  
30 assigned to the node  $K_1$  that has a leave field  $K_1$  assigned by the key distribution server that is, at this point, unknown to the user  $M_1$ . As described above, the leave field for a parent node can be calculated using the following formula:

Parent\_key = f(f(child\_key, seq\_n, pos#\_key) XOR offset)

The same algorithm can be used to generate the leave field of the node K1 from the secret key K\_m1 thus:

5

K1 = f[f(K\_m1, 1, 1) XOR offset\_m1]

Thus the key distribution server simply calculates the offset required to obtain K1 from the key K\_m1 and broadcasts that offset to the group. Of course, only user M<sub>1</sub> can use that offset to generate the leave field K1 because only user M<sub>1</sub> knows the key K\_m1.

The key server calculates the offset messages required by the user M<sub>1</sub> in order to generate, from the random key K\_m1, the leave field for each node in the hierarchical tree from the user's leaf node to the root of the tree i.e. the leave fields K1, K12 and K14 as follows:

20 K1 = f[f(K\_m1, 1, 1) XOR offset\_m1]  
 K12 = f[f(K1, 1, 12) XOR offset\_1]  
 K14 = f[f(K12, 1, 14) XOR offset\_12]

Once the new user has the leave key of the root node (the leave field of the session key) he can decrypt the group traffic.

The generation of keys K1, K12 and K14 by the user M<sub>1</sub> is represented diagrammatically in Figure 8. Figure 8 includes elements 2, 6, 8, 12, 14 and 18 each representing a one-way function f and elements 4, 10 and 16 each representing an XOR function and includes inputs K\_m1, offset\_m1, offset\_1, offset\_12 and generates outputs K1, K12 and K14. Input K\_m1 is connected to the input of one-way function 2. The output

of one-way element 2 is connected to a first input to XOR element 4, the second input to XOR element 4 being connected to input offset\_m1. The output of XOR element 4 is connected to the input of one-way function 6. The output of one-way function 6 provides the output K1 and is also connected to the input of one-way function 8. The output of one-way function 8 is connected to a first input of XOR element 10, the second input of XOR element 10 being connected to the input offset\_1. The output of XOR element 10 is connected to the input of one-way function 12. The output of one-way function 12 provides the output K12 and is also connected to the input of one-way function 14. The output of one-way function 14 is connected to a first input of XOR element 16, the second input of XOR element 16 being connected to the input offset\_12. The output of XOR element 16 is connected to the input of one-way function 18. The output of one-way function 18 provides the output K14.

In a similar manner, user  $M_2$  shares a random key  $K_{m2}$  with the server and receives offsets offset\_m2, offset\_2 and offset\_12 from the key server, user  $M_3$  shares a random key  $K_{m3}$  with the server and receives offsets offset\_m3, offset\_3 and offset\_34 from the key server and user  $M_4$  shares a random key  $K_{m4}$  with the server and receives offsets offset\_m4, offset\_4 and offset\_34 from the key server. Users  $M_2$ ,  $M_3$  and  $M_4$  then generate keys K2, K3 and K4 as follows:

$$K2 = f[f(K_{m2}, 1, 2) \text{ XOR offset\_m2}]$$

$$K3 = f[f(K_{m3}, 1, 3) \text{ XOR offset\_m3}]$$

$$K4 = f[f(K_{m4}, 1, 4) \text{ XOR offset\_m4}]$$

The users  $M_2$ ,  $M_3$  and  $M_4$  then generate the remaining leave keys for their branch of the tree as outlined above.

As outlined above, the join field is common to all members. The join field is modified each time that a new user joins the group. When a new user joins the group, there is no need to update any of the leave fields since the new user does not  
5 know the previous join field and therefore does not know the previous session key with which previous data was encrypted. Backward security is therefore achieved without updating the leave field of the session key (or, indeed, the leave field of any other node).

10

If backward security is not a requirement, then there is no need to modify the join field when a new member joins the group.

15 As with the prior art hierarchical trees described above with reference to Figures 1 to 3, the leave field of the session key must be updated each time a user leaves the group. Indeed (given the relationship of the leave keys in the tree), every leave field that the leaving user knew must be  
20 updated to ensure that that user cannot calculate the new leave field of the session key, and therefore calculate the new session key. This is required to ensure that the system has forward security.

25 When a new member joins the group, the join key is updated using the following formula:

$$\text{New\_Join\_Key} = f(\text{Old\_Join\_Key}, N)$$

30 N is a sequence number and is typically limited in size, perhaps to four bits. The sequence number N may be broadcast to the existing users. If so, given that each of those users knows the old join key, each of those users can generate the new join key. Thus each of the existing users of the group

can generate the new join key without requiring the encryption of information by the key distribution server and without requiring secure connections between the key distribution server and the users. Non-members cannot  
5 generate the new join key from the information broadcast since they do not know the old join key.

In a preferred embodiment of the invention, the new join key is generated thus:

10

$$\text{New\_Join\_Key} = f[\text{old\_key XOR opad}, f(\text{old\_key XOR ipad}, N)]$$

where

15  $\text{ipad} = \text{the byte } 0x36 \text{ repeated } B \text{ times}$   
 $\text{opad} = \text{the byte } 0x5C \text{ repeated } B \text{ times}$

N may be cyclic in which case it would be possible for a user to recover and catch up having missed a join key update  
20 instruction. Thus, if the key distribution broadcasts a join event with the sequence number  $N=8$  and a particular user had believed that the current sequence number was 6, that user can calculate the correct new join key from his current join key thus:

25

$$\text{Join\_key}(N=8) = f(f(\text{Join\_key}(N=6), 6), 7)$$

A new member of the group that is given the new join key cannot determine earlier join keys and hence cannot determine  
30 earlier session keys with which data has been transmitted (since the former sessions keys each include former join keys that are unknown to the new user). Thus the backward security of the communication system is ensured.

Similarly, whilst a former member of the group can calculate new join keys from the broadcast of sequence numbers, they do not know the new leave keys and hence cannot determine the new session keys. Thus the forward security of the  
5 communications system is ensured.

Thus, when a new member joins the group, the existing users generate the new join key themselves and the leave keys are unchanged. The new user must be sent the new join key  
10 together with all the leave keys between the leaf node of the new user and the root node in a secure manner (using a secret key shared with the key distribution server as described above). For a balanced binary hierarchical tree, each new user will be sent  $\log_2(N) + 1$  leave keys and 1 join key.  
15

Figure 9 shows how the hierarchical tree of Figure 6 is amended by the inclusion of a new member  $M_5$ . The new member  $M_5$  is associated with a node  $K_5$  and is made a sibling of node  $K_4$  with which user  $M_4$  is associated. A new parent node  $K_{45}$  is  
20 created for nodes  $K_4$  and  $K_5$ . The new node  $K_{45}$  takes the place in the tree that was previously allocated to node  $K_4$ .

Figure 10 lists, in the form of a flow chart, the steps required to integrate a new user is into the system. Those  
25 steps are:

1. "Group Access Request", step 20. The new member  $M_5$  contacts the key distribution server to request access to the group.  
30
2. "KDS Grants Access?", decision step 22. The key distribution server (KDS) decides whether or not the new member should be admitted to the group.

3. "New Member Assigned Node", step 24. If the key distribution server admits the new member access to the group, the key distribution server assigns a node to that user and updates its copy of the tree.

5

4. "Offset Messages Sent to New Member", step 26. The new member is sent all of the information that is required to gain access to the session key. This information can be sent using a reliable unicast protocol. At the same time, the key server updates the join key of the group.

10

In the example of Figure 9, the new member  $M_5$  requires the following information: key  $K_{m5}$ ; offset messages  $offset_{m5}$ ,  $offset_{45}$  and  $offset_{35}$ ; and sequence numbers  $n_{m5}$ ,  $n_{45}$  and  $n_{35}$ .

15

The key  $K_{m5}$  is shared between the key server and the new user  $M_5$  with a secure protocol. The key could be generated by either the key server or the new user.

20

5. "Leave field calculations", step 28. The new user  $M_5$  generates the leave keys from the leaf node  $K_5$  to the root node  $K_{15}$  as follows:

25  $K_5 = f[f(K_{m5}, n_{m5}, 5) \text{ XOR } offset_{m5}]$   
 $K_{45} = f[f(K_5, n_5, 45) \text{ XOR } offset_5]$   
 $K_{35} = f[f(K_{45}, n_{45}, 35) \text{ XOR } offset_{45}]$   
 $K_{15} = f[f(K_{35}, n_{35}, 15) \text{ XOR } offset_{35}]$

30 6. "Protocol Message to Existing Members", step 30. A protocol message is broadcast to the users in the group to inform them that another user has joined the group. The protocol message comprises a sequence number  $N$  and a position number. In this case, the position number is  $K_5$  (the



position of the new member  $M_5$ ) and K45 (the position of the nodes that needs to be added to the tree structure).

Thus the protocol message is: "N, POS: K5, K45".

5

7. "Update Join Field", step 32. The existing members of the group generate the new join key in the manner described above where the new join key is given by:

10  $f(\text{old\_join\_key}, N)$

8. "New Parent Node Generation", step 34. Member  $M_4$  (the sibling of the new member  $M_5$  in the hierarchical structure) generates new node K45 that is the parent of both nodes K4  
15 and K5. All of the information required by the member  $M_4$  to realise that a new node is required and to generate that node is contained in the protocol message described in step 6 above. The leave key for the new parent node K45 is generated thus:

20

$K45 = f[f(K4, n, 45)]$

The key K45 has already been calculated by the key distribution server (in the same manner) and used to generate  
25 the appropriate offset to send to the new user  $M_5$  to generate the key K45.

After the new parent node has been generated, the new member join event is complete and the event terminates at "End" step  
30 36. If the key distribution server denied the new member access to the group in the "KDS Grant Access?" decision box 22, then the join event is terminated at step 36 at this stage.

Thus the join event of the present invention is a very low cost operation in terms of the operations performed by the key distribution server, the data transmitted by the key distribution server and the computational effort required of  
5 the users.

The join event described above assumes the occurrence of a single join event. The present invention is also applicable to communication systems in which multiple simultaneous join  
10 events are allowed. The principal difference between a single join event and a multiple join event is that existing members of the system may be required to move down several layers of the hierarchical tree in which case they would have to generate several new tree nodes. The join message issued  
15 by the key distribution server for a multiple join event includes the sequence number N and the position of all the new nodes that are required to be generated.

Figure 11 shows how the final hierarchical tree of Figure 9  
20 is amended by the deletion of member  $M_4$ . When  $M_4$  leaves the group, node K45 is deleted and node K5 is promoted in its place. Nodes K5, K35 and K15 are then re-keyed to give  $K5'$ ,  $K35'$  and  $K15'$ .

25 Figure 12 lists, in the form of a flow chart, the steps required to remove a member from the system. Those steps are:

1. "Member Leave Instructions", step 38. An instruction to  
30 remove a member from the group is generated. This may take the form of a request from the user concerned (a voluntary removal) or the user in question may be ejected (a forced removal).

2. "Parent node deletion", step 40. The parent of the node associated with the member to be removed is deleted by the key distribution server from its tree.
- 5 3. "Sibling node promotion", step 42. The sibling of the node associated with the member to be removed is promoted by the server to the position in the tree stored by the server that was previously occupied by the node deleted in step 2.
- 10 4. "New Leave Keys Generated By KDS", step 44. As noted above, all of the leave keys known to the user being removed from the group should be updated to ensure the forward security of group communication. The key distribution server (KDS) generates these new keys.
- 15 5. "Protocol Message To Remaining Members", step 46. A protocol message is broadcast to all remaining users in the group. The protocol message comprises the node that is associated with the member that is leaving the group
- 20 (#Position Node Leaving Tree), the parent node deleted in step 2 above (#Position Node Removed) and the offset messages required by the users to calculate the new leave fields (Offset\_#Pos) where '#Pos' refers to the node to which the offset must be applied so that Offset\_3 is the offset
- 25 required to generate the leave key of the parent of node 3.

Thus the protocol message is: "#Position Node Leaving Tree; #Position Node Removed; Offset\_#Pos".

- 30 6. "New Leave Key Calculations", step 48. The remaining members calculate the updated leave keys. The new leave key K5' is derived by both the key distribution server and user  $M_5$  using a single one-way function thus:

$K5' = f(K5, \text{Sequence number}, 5)$

The new key  $K5'$  is used to generate the new keys  $K35'$  and  $K15'$ . The new leave keys are defined by the formulae:

5

$K35' = f[f(K3, \text{sequence number}, 35) \text{ XOR } \text{Offset\_3}]$

$K15' = f[f(K12, \text{sequence number}, 15) \text{ XOR } \text{Offset\_12}]$

$K15' = f[f(K35', \text{sequence number}, 15) \text{ XOR } \text{Offset\_35'}]$

10 where:

$\text{Offset\_3} = f(K3, \text{sequence number}, 35) \text{ XOR } f(K5', \text{sequence number}, 35)$

15  $\text{Offset\_35'} = f(K35', \text{sequence number}, 15) \text{ XOR } f(K12, \text{sequence number}, 15)$

$\text{Offset\_12} = f(K12, \text{sequence number}, 15) \text{ XOR } f(K35', \text{sequence number}, 15).$

It is possible that a user may miss a protocol message. In  
20 such circumstances, that user will not have updated the tree  
(or that branch portion that the user keeps) and accordingly  
will not know the new session key. The user will not then be  
able to decrypt the information transferred by the system.

25 To deal with such circumstances, the key distribution server  
may provide hint messages to enable users rekey the tree in  
the event that they have missed a protocol message. In one  
embodiment of the invention, the hint message takes the form  
of the protocol message that is attached a data packet.

30 Simply attaching the previous protocol message to each data  
packet is feasible since each protocol message is likely to  
be relatively small when compared to the size of a data  
packet.

A user that has missed a protocol message will be able to identify this since he will not be able to decrypt the data transferred by the system. That user will simply need to extract the protocol message from the data stream.

5

The system could be extended to provide a number of the most recent protocol messages so that misses of several protocol messages can be caught up. The number of protocol messages that can be attached to the data messages is only limited by the amount of data bandwidth the designer of the system is willing to allocate to the hint messages.

The user of hint messages is well suited to the key distribution system of the present invention since the key update information is not encrypted, and thus can be shared across all members. There is therefore no bar to transmitted one or more hint messages with the data packets.

The leave event described above assumes the occurrence of a single leave event. The present invention is also applicable to communication systems in which multiple simultaneous leave events are allowed. The principal difference between a single leave event and a multiple leave event is that the removal of several users leaves the possibility of the remaining tree being configured in more than one possible way. Thus, in such situations the key distribution server must take decisions about the path from each remaining user to the root node and issue protocol messages accordingly.

Thus it can be seen that the OHBT protocol provides a system in which when a new user joins the group, only a sequence number N needs to be broadcast to the group (although, as discussed above, it is possible for a user to recover, even if one or more protocol messages are missed). Thus, the join

event scales extremely efficiently as the number of users increases. Further, each user generates the new join key using a straightforward algorithm. Thus the computational requirements of the users for a join operation is low. A  
5 join event is secure because it is computationally infeasible for the new member to calculate the previous join key with a finite probability.

The leave operation requires the calculation and broadcast of  
10 offset messages. These messages can be broadcast as plain text messages and do not need to be encrypted. The users generate the new leave keys using a simple double one-way function. Thus, the computational requirements of the users for a leave operation is also low. A leave event is secure  
15 because although the former member knows the join key and the previous leave key for the root node, he cannot generate the new leave key of the root node and therefore cannot generate the new session key.

20 In general it is anticipated that the number of join events will be greater than the number of leave events for most systems. This is because there is likely to be a multiple leave event at the end of the communication session. Accordingly, the use of a low cost join event as in the  
25 present invention is an advantage.

As the number of users increases ever further, it may be advantageous to provide more than one key distribution server or to distribute the function of the key distribution server  
30 among a number of nodes. The use of multiple key distribution servers present issues of synchronisation between the servers. The present invention assists in the transfer of information between key distribution servers since key information in the present invention is transmitted

as plain text messages. The lack of encryption for the transmission of key information simplifies at least some of the issues associated with the use of multiple key distribution servers.

5

The description of the invention given above assumes that the key nodes are arranged in a binary tree. A binary tree is not a requirement of the invention. The key nodes could be arranged in a P-ary tree (where P is greater than or equal to 10 2), for example.

The key distribution sever is preferably centralised at one location. Often it will be convenient for this to be at the 15 source of application data, e.g. a video stream. Arrangements in which the functions of the key distribution server are duplicated and/or distributed over several machines at difference locations.